High Availability con Linux

Enrico Cavalli
CILEA, Segrate

Abstract
Linux si qualifica sempre più come sistema operativo per l'enterprise e si slega dal mondo accademico, dove la sperimentazione e il gusto per l'innovazione spesso vanno contro la necessità di stabilità e affidabilità. Per un'azienda l'affidabilità dei sistemi informativi è invece critica. Vediamo come progettare un sistema altamente affidabile con Linux.

Keywords: Telematica, Cluster, High Availability, Linux, Heartbeat, Keepalived, VRRP.

Premessa

Progettare sistemi altamente affidabili è il sogno o forse l'utopia di ogni persona che lavori nel campo dell'informatica. Avere a disposizione sistemi affidabili è senza dubbio l'aspettativa degli utenti. Linux sta prendendo sempre più piede nel cosiddetto ambito *enterprise*, ovvero nell'azienda, piccola o grande che sia. In certe realtà economiche l'affidabilità di un sistema informativo è di fondamentale importanza, perché un disservizio di qualsiasi natura si traduce inesorabilmente in danni economici o di immagine. Di qui la necessità dell'*high availability*, ovvero di sistemi che siano in grado di raggiungere l'agognato 99.999% di disponibilità.
Se i fantomatici cinque o più nove sono forse un traguardo troppo ambizioso per macchine la cui architettura è la stessa dei PC sulle nostre scrivanie e nelle nostre case, d'altra parte possiamo provare ad avvicinarci a questa meta, con una tecnica che si traduce in una parola sola: *clustering*.

I cluster

Un *cluster* è sostanzialmente un gruppo di macchine che lavora in parallelo per fornire un servizio. Esistono sostanzialmente due architetture di *clustering*. La prima sfrutta la tecnica del *load-balancing*: in questo modello più macchine effettivamente si distribuiscono i carichi di lavoro e possono lavorare in concomitanza, con vari gradi di parallelismo, per svolgere un unico compito. Nella seconda architettura una macchina svolge il lavoro, mentre un'altra ha funzione di backup, restando in uno stato quiesciente (*hot-standby*) pronta a sostituire la prima in caso di un guasto o malfunzionamento di quest'ultima.
In realtà i due approcci spesso coesistono, specialmente nelle architetture più complesse ed affidabili. Pensiamo ad esmepio ad un servizio di web server altamente affidabile. Questo potrebbe essere realizzato con un cluster di *n* macchine in grado di fornire le pagine web in parallelo. Nel cluster avremo più macchine sostanzialmente identiche che condividono in qualche modo lo spazio disco contenente le pagine web. Naturalmente le richieste dovranno essere smistate alle varie macchine del cluster. Un oggetto che svolga questo compito di arbitraggio è detto load-balancer. A questo punto il bilanciatore diventerebbe un cosiddetto SPOF (Single Point Of Failure) dell'architettura. Un modo naturale per eliminare questo inconveniente è applicare tecniche di clustering anche per il load-balancer, affiancandolo con una macchina identica in hot-standy.
In figura 1 (pagina seguente) viene rappresentata schematicamente questa architettura.
Esistono vari prodotti più o meno costosi per realizzare dei cluster altamente affidabili con Linux, ma questa non è la sede per entrare nel merito delle varie offerte commerciali. Ci concentreremo invece su implementazioni open-source di clustering in ambito Linux.
In particolare analizzeremo due soluzioni per la creazione di un cluster di due macchine, di cui una in hot-standby. Affronteremo inoltre il caso più semplice possibile, ovvero l'alta affi-

dabilità di un firewall, dove sostanzialmente non è necessario avere uno storage condiviso per i dati. In effetti in questo caso basta replicare opportunamente sulle macchine coinvolte i pochi file di configurazione delle regole di firewall e degli applicativi che si occupano dell'high-availability.

Risorse e battiti: *heartbeat*

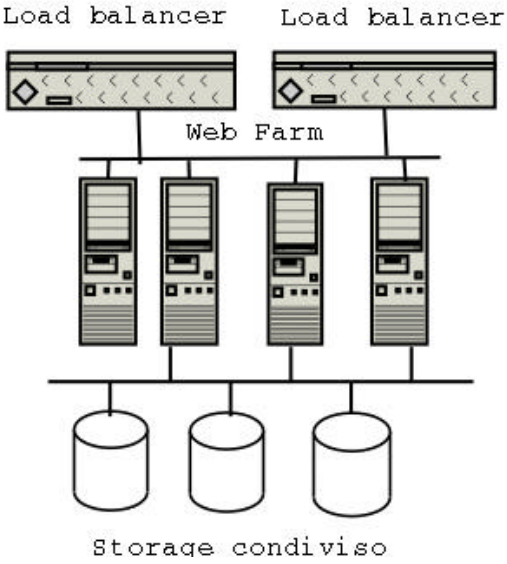
Uno dei software più semplici per realizzare questo tipo di cluster è *heartbeat* [1]. L'idea alla base di questo software è un monitoraggio reciproco delle macchine coinvolte nell'architettura. Queste emettono dei segnali periodici che vengono interpretati dal partner come segno di un buono stato di salute della controparte. Questa sorta di battito cardiaco può viaggiare su diversi canali: broadcast udp su ethernet, multicast sempre su ethernet, oppure come segnali su linea seriale dedicata. La configurazione è piuttosto banale e si traduce in poche direttive nel file di configurazione principale di heartbeat (`/etc/ha.d/ha.cf`) che deve essere identico su entrambe le macchine. Ad esempio le righe:

```
serial /dev/ttyS0
udp eth1
udpport 694
```

istruiscono hearbeat ad emettere segnali sulla seriale /dev/ttyS0, e dei broadcast sulla porta 694/UDP attraverso la scheda ethernet eth1. In questo file dovremo anche indicare quali sono le macchine che partecipano al cluster, indicandole con i rispettivi hostname con le seguenti direttive:

```
node firewall-node0.domain.com
node firewall-node1.domain.com
```

Esiste poi un secondo file (/etc/ha.d/haresources) dove vengono configurate le risorse del cluster: nel linguaggio di heartbeat una risorsa è un servizio che deve essere reso altamente disponibile. Nel nostro caso abbiamo a che fare con un firewall, quindi le uniche risorse che il cluster deve mantenere disponibili sono indirizzi IP. Da un lato avremo ad esempio l'indirizzo IP che le macchine sulla LAN utilizzano come default gateway, dall'altro avremo l'indirizzo con cui dobbiamo effettuare il NAT delle macchine sulla LAN. Ovviamente come risorsa potremmo avere avere anche gli indirizzi pubblici che corrispondono ad indirizzi di macchine in realtà poste in una DMZ su una rete privata.



Il tutto si ridurebbe ad un riga nel file delle risorse del cluster:

```
firewall-node0.domain.com
192.168.1.25 192.168.2.254
131.175.1.135
```

Figura 1 - Schema di webfarm con bilanciatori di carico.

Facciamo un esempio concreto per meglio comprendere come funziona il tutto. Supponiamo che il nostro provider ci assegni un certo numero di indirizzi pubblici, ad esempio la rete 131.175.1.0/24. Supponiamo inoltre di assegnare ai PC sulla nostra LAN indirizzi della rete 192.168.1.0/24 e di scegliere indirizzi 192.168.2.0/24 per i server posti nella DMZ. Supponiamo inoltre di mascherare con l'indirizzo 131.175.1.1 i client della LAN, e di avere un web server sull'indirizzo 131.175.1.2. Avremo quindi bisogno di due macchine con tre schede di rete, alle quali assegneremo degli indirizzi scelti tra quelli disponibili nelle rispettive reti. Questi indirizzi **non** devono assolutamente essere quelli che vogliamo mantere altamente disponibili con il cluster.
Questi ultimi dovranno rispettivamente essere:

1. l'indirizzo con cui i PC della LAN vengono mascherati, ovvero 131.175.1.1;

2. l'indirizzo pubblico del web server (131.175.1.2), che tramite opportuno NAT tradurremo in un indirizzo della DMZ privata;
3. l'indirizzo che i server sulla DMZ usano come default gateway, ad esempio 192.168.2.254;
4. l'indirizzo che i client della LAN usano come default gateway, ad esempio 192.168.1.254.

Il file di configurazione delle risorse assume allora il seguente aspetto:

```
firewall-node0.domain.com
192.168.1.254          192.168.2.254
131.175.1.1 131.175.1.2
```

Si tratta di un'unica riga, anche se qui è spezzata per esigenze tipografiche.
Osserviamo due cose riguardo a questa direttiva di haresources:

1. firewall-node0.domain.com oltre ovviamente a dover essere dichiarato come nodo in ha.cf, indica il nodo su cui le risorse devono normalmente essere disponibili: in pratica indichiamo il nodo master, mentre l'altro resta in hot-standby;
2. non è necessario specificare su quali interfacce devono essere impostati i vari indirizzi: heartbeat è in grado di determinarlo autonomamente basandosi sull'attuale configurazione delle interfacce e sulle regole di routing (si riamanda alla documentazione sul sito in merito a questo secondo punto; si tenga solo presente che salvo casi molto particolari non è necessario impostare configurazioni complicate).

Non intendiamo entrare nel merito dell'effettiva configurazione del firewall per un'architettura di questo tipo, con LAN e DMZ. Precisiamo solamente che presupponiamo che il firewall venga attivato su entrambi i nodi del cluster alla partenza delle macchine, ovviamente prima della partenza di heartbeat.
La domanda a questo punto potrebbe essere: come fa a funzionare il tutto e soprattutto come si raggiunge l'alta disponibilità?
Per capirlo caliamoci ad esempio nei panni di un client sulla LAN. Come client so che devo contattare il router 192.168.1.254 per destinazioni che non si trovano sulla mia rete locale. Questo indirizzo viene configurato da hearbeat come un alias sull'opportuna interfaccia del nodo 0 del cluster. Se per qualche motivo il nodo 0 dovesse smettere di funzionare, heartbeat provvederebbe a:

1. assegnare l'indirizzo al nodo1 del cluster;
2. informare i client della LAN che è cambiato il MAC address corrsipondente all'indirizzo IP 192.168.1.254.

Se la prima parte di questo processo, denominato failover, è semplicemente banale (il nodo di backup effettua sostanzialmente degli ifconfig con gli indirizzi opportuni), è opportuno precisare meglio cosa c'è dietro alla seconda parte. La tecnica utilizzata da heartbeat è quella di spedire in modalità broadcast delle risposte ARP non sollecitate, ovvero dei gratuitous ARP. In pratica vengono spediti alcuni pacchetti che informano tutte le macchine della rete che un determinato indirizzo IP è associato ad un determinato MAC address, che ovviamente non è più quello del nodo principale. Grazie a questi ARP gratuiti i vari dispositvi in rete hanno l'opportunità di modificare opportunamente la propria ARP table.
Facciamo notare che la tecnica degli ARP gratuiti può in alcuni casi essere utilizzata per operazioni illecite come lo sniffing su reti switchate. Per questo motivo esistono router e switch che ignorano, o possono essere configurati per ignorare, gli ARP gratuiti. Ovviamente per il funzionamento di heartbeat è importante verificare di non trovarsi in questa condizione. Per concludere, ribadiamo che le macchine partecipanti al cluster dovranno essere sincronizzate tra di loro. Nel nostro caso basta una copia periodica dei file di configurazione del firewall e di hearbeat, senza la necessità di disporre di sistemi di storage condivisi, necessari invece per architetture più complesse.

Possibili problematiche

Il sistema descritto funziona egregiamente nei casi in cui siano necessari interventi programmati sul nodo principale del cluster. In questo caso il sistemista può spegnere la macchina con una consueta procedura di shutdown ed heartbeat è in grado di gestire perfettamente il failover come descritto poco fa. Questa potrebbe essere una cosa di non poco conto per quelle aziende che utilizzano un firewall, in quanto consente la programmazione degli interventi negli orari più comodi, senza avere impatti negativi sul funzionamento della propria rete. L'architettura si comporta molto bene anche in caso di crash completi (ad

esempio un alimentatore che si rompa) del nodo principale del cluster: se questo non emette battiti per un certo lasso di tempo configurabile dall'amministratore, la seconda macchina subentra in maniera trasparente.
Esistono però tipi di guasto in cui questa architettura non si comporta nel modo sperato. Supponiamo ad esempio che si guasti un'interfaccia di rete sulla quale non è in ascolto heartbeat. A tutti gli effetti il firewall non sta funzionando, ma heartbeat non interviene in quanto ciascun nodo continua a sentire "vivo" il proprio compagno.
Ovviamente in un caso come questo basterebbe un intervento di un tecnico che spenga la macchina: quella di backup interverrebbe subito.
Certo è che non si tratta di una procedura che scatta automaticamente, ma è pur sempre vero che avere già pronta una macchina gemella in grado di sotituirsi a quella con la scheda rotta consentirebbe di dedicarsi con calma al problema dell'altra.
Non si deve pensare che attivare heartbeat su tutte le schede di rete del firewall risolva il problema. Secondo la logica di heartbeat infatti, la controparte viene dichiarata "morta" se e solo se non viene percepito più battito su tutti i canali configurati.
Si tratta di una scelta implementativa, che può avere senso in alcuni casi, meno in altri. Può essere una scelta criticabile, ma ha le sue motivazioni, soprattutto perché heartbeat affonda le sue radici nel clustering di database server.
Per l'high availability di un firewall sicuramente non è il modo migliore di affrontare il problema.

Keepalived: un'implementazione di VRRP

Il secondo prodotto che descriviamo si chiama *keepalived* [2] (fig. 2).
Questo è un framework molto complesso per l'implementazione di cluster altamente affidabili con Linux. Il software è integrato con LVS [3] (Linux Virtual Server), che implementa vari criteri per progettare load-balancer per server farm.
Tra le molte cose che è in grado di fare, keepalived serve principalmente per configurare in maniera semplice un load-balancer e per monitorare le risorse all'interno della server farm.
Un bilanciatore infatti, non solo deve inoltrare le richieste degli utenti alle macchine della server farm, ma deve anche evitare di inoltrarle a macchine che per qualisasi motivo non siano in grado di fornire il servizio in quel momento, altrimenti non si otterrebbe l'alta affidabilità.
Keepalived è estremamente semplice da configurare nel caso dell'high-availability di un firewall.
L'idea è per certi versi simile, ma allo stesso tempo diversa rispetto ad heartbeat.

Figura 2 - Logo di Keepalived

Keepalived implementa infatti VRRP (Virtual Router Redundancy Protocol), un protocollo tramite il quale più router di una LAN possono dinamicamente assegnarsi la responsabilità di rispondere a determinati indirizzi IP. Ricordiamo infatti che per l'alta affidabilità di un firewall (che è anche un router in un certo senso), è sufficiente mantenere l'alta disponibilità di indirizzi IP, siano essi i default gateway dei client di una LAN, oppure gli indirizzi pubblici che vengono tradotti in indirizzi privati con regole di NAT. Questi IP possono in un certo senso essere virtualizzati, e passare da un firewall ad un altro che implementi VRRP. Mentre con heartbeat tutte le macchine partecipanti ad un cluster trasmettono i propri segnali, con VRRP solo il nodo MASTER trasmette su un canale multicast dedicato (vrrp.mcast.net ovvero 224.0.0.18) che gli altri router hanno l'obbligo di non inoltrare verso altre interfacce secondo l'RFC 2338 [4]. In parole povere questi segnali devono restare confinati nella LAN di provenienza. I nodi di backup restano in ascolto su questo canale, senza trasmettere. Nel momento in cui il MASTER dovesse terminare la trasmissione, il nodo di BACKUP interverrebbe, assumendone l'IP virtuale. Se avessimo più nodi di backup assegne-

remmo priorità diverse a ciascuno: quello a priorità più alta subentrerebbe al MASTER, mentre gli altri rimarrebbero comunque nello stato di standby, pronti ad intervenire in caso di caduta anche del neonato detentore del Virutal IP.
Anche in questo caso la configurazione è semplicissima, e si limita al file: `/etc/keepalived/keepalived.conf`. Vediamo come apparirebbe tale file nell'esempio già trattato per heartbeat, limitandoci solo alle impostazioni essenziali e rimandando il lettore al sito di keepalived [2].

```
vrrp_sync_group {

      VI_0
      VI_1
      VI_2

}

vrrp_instance VI_0 {
      state MASTER
      interface eth0
      virtual_router_id 100
      priority 100
      virtual_ipaddress {
       131.175.1.1
       131.175.1.2
      }
}

vrrp_instance VI_1 {
      state MASTER
      interface eth1
      virtual_router_id 100
      priority 100
      virtual_ipaddress {
       192.168.2.254
      }

vrrp_instance VI_2 {
      state MASTER
      interface eth2
      virtual_router_id 100
      priority 100
      virtual_ipaddress {
       192.168.1.254
      }
}
```

Ovviamente sul nodo (o sui nodi) di backup le istanze di VRRP saranno indicate con "`state BACKUP`" e avranno una priorità decrescente. Facciamo notare come su ogni segmento di LAN sia possibile far coesistere più router "virtuali", perché ciascuno avrà il proprio virtual_router_id. Possiamo configurare fino a 255 router virtuali secondo lo standard IETF (RFC 2338 [4]), e un numero praticamente arbitrario con keepalived che si discosta dallo standard non implementando il virtual MAC address (VMAC).
Secondo la RFC infatti, un router che implementi VRRP deve rispondere alle richieste ARP per un indirizzo IP virtuale con una MAC address nella forma `00-00-5E-00-01-{VRID}`, dove VRID è per l'appunto l'identificativo del router virtuale. Lo standard prevede esplicitamente che il router non risponda con il MAC address dell'interfaccia fisica sottostante. keepalived al contrario risponde con il MAC address dell'interfaccia fisica. Questo comportamento è dettato dal fatto che Linux può impostare arbitrariamente il MAC address delle schede di rete, ma non è in grado di associare più di un MAC address ad ogni interfaccia fisica. Se si vuole supportare più di un virtual_router_id per interfaccia, è necessario quindi discostarsi dallo standard e utilizzare invece la tecnica degli ARP gratuiti, proprio come heartbeat.
Ritorniamo ora ad analizzare la configurazione sopra riportata, e soffermiamoci sulla direttiva forse più importante, vrrp_sync_group. Questa permette di trattare come un'unica entità più IP virtuali su schede di rete diverse: questo risolve il problema di heartbeat cui accennavamo in precedenza, relativo al guasto di una singola scheda di rete. Con keepalived, al contrario, un guasto su una singola scheda di rete fa perdere al sistema tutti i virtual IP, in modo che un nodo di backup possa correttamente subentrare al MASTER non più in grado di erogare il servizio.
Aggiungiamo che, tra l'altro, keepalived implementa anche dei controlli sulla presenza del link fisico sulle interfacce di rete. Con keepalived, se stacchiamo **uno solo** dei cavi di rete, il router di BACKUP subentra al MASTER. Con heartbeat questo non accadrebbe, perché il nodo di backup si sotituisce al principale se e solo se non sente pià battiti su **tutti** i canali configurati, schede di rete e cavi seriali.
Nonostante keepalived appaia sicuramente più completo di heartbeat, è doveroso segnalare che si tratta di un prodotto più giovane di quest'ultimo, e non completamente maturo dal

punto di vista dell'autenticazione dei messaggi trasmessi sul canale multicast – anche se confidiamo che presto gli sviluppatori completeranno questa parte. A sviluppo ultimato keepalived offrirà maggiori garanzie dal punto di vista della sicurezza, in quanto utilizza il protocollo AH (Authentication Header) per l'autenticazione dei messaggi multicast inviati.
Inoltre keepalived è un framework estremamente complesso, nel quale l'implementazione di VRRP è solo un componente. Con queste osservazioni vogliamo precisare che keepalived non deve necessariamente essere considerato migliore o peggiore rispetto ad heartbeat: sono due prodotti con obiettivi diversi.
Confidiamo di aver dato dettagli sufficientemente obiettivi, che consentano di effettuare valutazioni e test in maniera autonoma per scegliere il prodotto più consono alle proprie esigenze.

Management

La potenza è nulla senza il controllo, recita un famoso slogan pubblicitario che ben si può applicare alle considerazioni che stiamo per affrontare. Se da un lato abbiamo visto due ottimi prodotti per l'high- availability, dobbiamo purtroppo segnalare che il panorama di strumenti software per il management di cluster di firewall Linux non sono altrettanto completi. Sarebbe infatti utile poter disporre di un'interfaccia, magari anche grafica, per gestire un cluster. Un'interfaccia di questo tipo dovrebbe consentire il management dei filtri del firewall, oltre a consentire una facile configurazione del software di clustering che si è deciso di adottare. Allo stato attuale non conosciamo prodotti che affrontino la questione in modo completo e sufficientemente personalizzabile da parte di un amministratore esperto.
Se dovessimo dare un suggerimento sulla strada che ci sembra più sensata da percorre, consiglieremmo l'adozione di *cfengine* [5] come motore per l'implementazione delle configurazioni sulle macchine coinvolte nel cluster. Sopra a cfengine sarebbe utile sviluppare un'interfaccia – magari web – per la configurazione del tutto. Possiamo infine segnalare un ottimo prodotto open-source per scrivere regole di firewall: fwbuilder [6]. Si tratta di un programma che gira sotto X Window, ed è molto simile come aspetto grafico all'interfaccia di Checkpoint FW1.
Per completezza ed obiettività segnaliamo che, dal punto di vista del management, l'offerta commerciale è, per ovvi motivi, sicuramente più variegata e completa. Con questo non intendiamo indirizzare verso soluzioni commerciali. Al contrario speriamo di incentivare la realizzazione di soluzioni open-source.

Possibili sviluppi

L'appetito vien mangiando, si sa. Visto che abbiamo imparato a costruire un cluster di firewall, vediamo ora cosa possiamo mettere dietro un firewall che sia anche un bilanciatore di carico magari per una webfarm, anche se tratteremo solo sommariamente la questione. Innanzitutto cerchiamo di capire meglio che cosa sia una web-farm, e perché mai dovremmo implementarne una. Una web-farm è un cluster di macchine che lavorano in parallelo servendo oggetti web ai clienti che ne facciano richiesta. Un cluster di macchine, offrendo un'alta ridondanza dei singoli componenti, risulta sicuramente più affidabile di una singola macchina, ma non solo: è sicuramente più scalabile. Se un domani il sito avesse un successo tale per cui il nostro cluster non riuscisse più a soddisfare tutte le richieste, basterebbe aggiungere nodi al cluster. Ovviamente quello del sito web è solo un esempio: possiamo adottare la soluzione di farm per molteplici servizi di rete.
Precisiamo subito che le macchine facenti parte della farm non devono necessariamente essere Linux, ma possono essere basate su un sistema operativo qualsiasi. Certamente è molto comodo mettere un cluster Linux di fronte alla farm, con la funzione di firewall e di bilanciamento dei carichi di lavoro. Il software utilizzabile a questo scopo è Linux Virutal Server [3], come abbiamo già avuto modo di vedere in precedenza. Possiamo scegliere di configurare "a mano" il bilanciatore, oppure possiamo utilizzare keepalived per la configurazione e il monitoraggio dei componenti della farm. Vediamo come funziona in pratica il bilanciamento, ovvero che cosa è in grado di fare Linux Virtual Server. Innanzitutto il servizio di rete che vogliamo rendere altamente disponibile e scalabile verrà registrato su un indirizzo IP che manterremo in high-availability grazie alle tecniche illustrate in precedenza.

Sulle due macchine del cluster per il bilanciamento dei carichi installeremo LVS. Quando una richiesta arriva al nostro bilanciatore, questo la smisterà ad un'opportuna macchina della farm. Il bilanciamento vero e proprio può avvenire in tre modi:

1. tramite NAT
2. tramite IP tunnelling
3. tramite Direct Routing

Nel primo caso la farm può essere costituita da qualsiasi macchina supporti il protocollo TCP/IP, quindi non necessariamente Linux. Il load-balancer in questo caso riscrive i pacchetti IP cambiando l'indirizzo IP destinazione con quello di una delle macchine nella farm. Queste ultime devono avere come default gateway il load-balancer stesso (o meglio un opportuno IP virtuale) perché i pacchetti dovranno opportunamente essere ritradotti nel percorso inverso. Si intuisce chiaramente che in questo caso il bilanciatore può facilmente trasformarsi in un collo di bottiglia per quanto riguarda il throughput di rete.

Negli altri due casi i pacchetti IP ricevuti dal load-balancer vengono incapsulati in un tunnel IP e destinati ad una macchina della farm, oppure vengono riscritti e rispediti a livello 2, indirizzando direttamente il MAC address di una macchina della farm. La tecnica dell'IP tunnelling ci risulta certificata solo con Linux, ma non è detto che sia l'unico sistema operativo utilizzabile in questa modalità a bordo delle macchine della farm. La tencica del Routing Diretto può invece funzionare con qualsiasi sistema operativo, anzi Linux è quello più problematico da gestire in questo caso. Senza addentrarci nei dettagli dei due metodi, osserviamo però come in questi ultimi due casi le macchine della farm non debbano utilizzare il load-balancer come default gateway. Si crea quindi un routing asimmetrico: i pacchetti destinati alla farm passano per il load-balancer soltanto in ingresso, mentre in uscita le singole macchine della farm rispondono direttamente all'indirizzo IP del richiedente. Il vantaggio è che il load-balancer non è più un collo di bottiglia. LVS è in grado di adottare diversi schemi di scheduling per scegliere su quale macchina della farm assegnare le varie richieste: si va dal semplice round-robin, a schemi più sofisticati dove si tiene conto dell'effettivo carico di lavoro già affidato ad ogni macchina e della capacità elaborativa delle singole macchine della farm (weighted least-connection scheduling). Gli schemi tra cui scegliere sono molti, e sarà compito di chi progetta la farm scegliere quello più adeguato alle proprie esigenze. Indipendentemente dall'algoritmo di scheduling che si sceglie di adottare, LVS è in grado di gestire il problema della persistenza delle connessioni. Una volta che l'algoritmo ha assegnato un dato client ad una specifica macchina all'interno della farm, molto spesso è necessario che le future richieste vengano ancora indirizzate a quest'ultima, ovviamente se è ancora disponibile. Si pensi ad esempio a questioni legate alla gestione delle sessioni web. Con LVS è possibile configurare in maniera piuttosto fine quale tipo di persistenza delle connessioni debba essere adottato.

Per concludere osserviamo che LVS, a partire dalla versione 0.9.2, supporta ache la sincronizzazione delle connessioni tra due load-balancer in high-availability. La sincronizzazione tra le due macchine è molto importante in caso di guasto del nodo master: in questo modo il nodo di backup partirebbe avendo già quasi tutte le informazioni sulle connessioni in corso, penalizzando il meno possibile le applicazioni. Facciamo notare che questa sincronizzazione di stato non è attualmente presente nel codice di firewalling di Linux, anche se esistono progetti di sviluppo in questo senso. Un firewall con stateful inspection quale iptables è proprio un'applicazione molto sensibile da questo punto di vista: un failover non sarà mai completamente trasparente agli utenti senza sincronizzazione tra i nodi.

Conclusioni

L'alta affidabilità con sistemi Linux può facilmente diventare una realtà, con investimenti tutto sommato non considerevoli, visto che si tratta di implementare un congruo numero di macchine di architettura Intel, facilmente reperibili sul mercato a prezzi concorrenziali. Non va dimenticato che esiste comunque una considerevole scelta di prodotti commerciali per il clustering, che abbiamo esplicitamente taciuto in questo articolo. Un aspetto che differenzia sicuramente l'offerta commerciale da quella open-source è la presenza di strumenti più o meno sosfisticati per il management di un cluster (monitoring dello stato dei nodi, deployment centralizzato del software e relative configurazioni, aggiunta e dismissione di

nodi nel cluster e così via). Invitiamo quindi il lettore interessato a sperimentare le idee presentate e anche a visitare l'URL:

http://www.linux-ha.org/commercial.html

per rendersi conto direttamente del cospicuo numero di soluzioni di clustering in ambito Linux.

Bibliografia

[1] Linux-ha, URL: http://www.linux-ha.org
[2] Keepalived, URL:
http://www.keepalived.org
[3] Progetto Linux Virtual Server,
URL: http://www.linuxvirtualserver.org
[4] L'RFC del protocollo VRRP,
URL: http://www.ietf.org/rfc/rfc2338.txt
[5] Cfengine, URL:
http://www.iu.hio.no/cfengine/
[6] Fwbuilder, URL:
http://www.fwbuilder.org/